

#3

PATENT  
0630-1060P

IN THE U.S. PATENT AND TRADEMARK OFFICE

Applicant: Min-Seok JANG  
Appl. No.: NEW APPLICATION Group: UNKNOWN  
Filed: March 17, 2000 Examiner: UNASSIGNED  
For: METHOD FOR IMPLEMENTING EVENT TRANSFER  
SYSTEM OF REAL TIME OPERATING SYSTEM

JCS64 U.S. PTO  
09/528028  
03/17/00

L E T T E R

Assistant Commissioner for Patents  
Washington, DC 20231

March 17, 2000

Sir:

Under the provisions of 35 U.S.C. § 119 and 37 C.F.R. § 1.55(a), the applicant(s) hereby claim(s) the right of priority based on the following application(s):

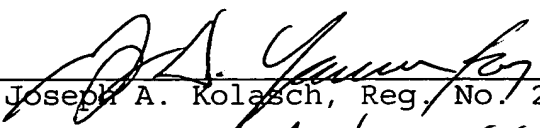
<u>Country</u>	<u>Application No.</u>	<u>Filed</u>
KOREA	9449/1999	March 19, 1999

A certified copy of the above-noted application(s) is(are) attached hereto.

If necessary, the Commissioner is hereby authorized in this, concurrent, and future replies, to charge payment or credit any overpayment to Deposit Account No. 02-2448 for any additional fee required under 37 C.F.R. §§ 1.16 or 1.17; particularly, extension of time fees.

Respectfully submitted,

BIRCH, STEWART, KOLASCH & BIRCH, LLP

By   
Joseph A. Kolasch, Reg. No. 22,463

P.O. Box 747  
Falls Church, VA 22040-0747  
(703) 205-8000  
By No. 35,416

JAK:jcp  
0630-1060P

Attachment

Min-Seok JANG  
"METHOD FOR IMPLEMENTING  
EVENT TRANSFER  
SYSTEM OF REAL  
TIME OPERATING SYSTEM"  
Filed 3-17-2000

Doc. No. 620-1060P  
BSKB  
(10) 205-8000

대한민국 특허청  
KOREAN INDUSTRIAL  
PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto  
is a true copy from the records of the Korean Industrial  
Property Office.

출원 번호 : 1999년 특허출원 제9449호  
Application Number

출원 년 월 일 : 1999년 3월 19일  
Date of Application

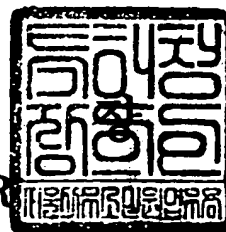
출원인 : 엘지전자 주식회사  
Applicant(s)



199 9 년 12월 13일

특 허 청

COMMISSIONER



CERTIFIED COPY OF  
PRIORITY DOCUMENT



23-1

**【수수료】**

【기본출원료】 20 면 29,000 원

【가산출원료】 3 면 3,000 원

【우선권주장료】 0 건 0 원

【심사청구료】 0 항 0 원

【합계】 32,000 원

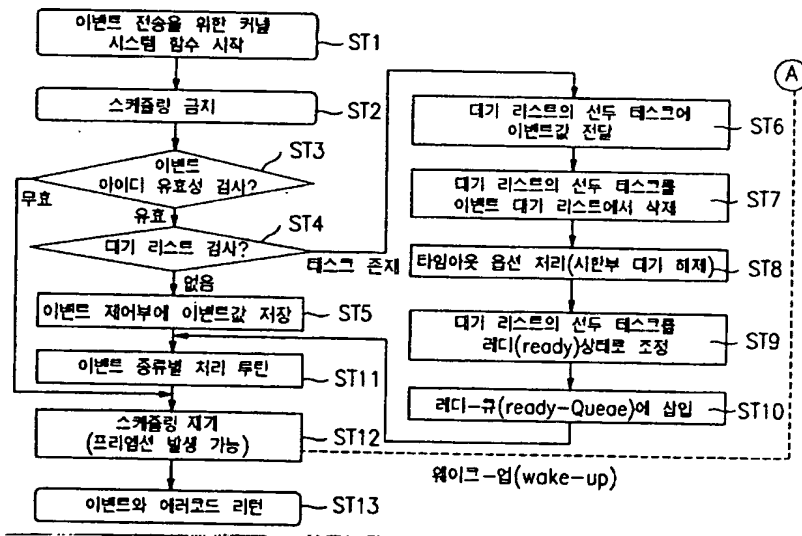
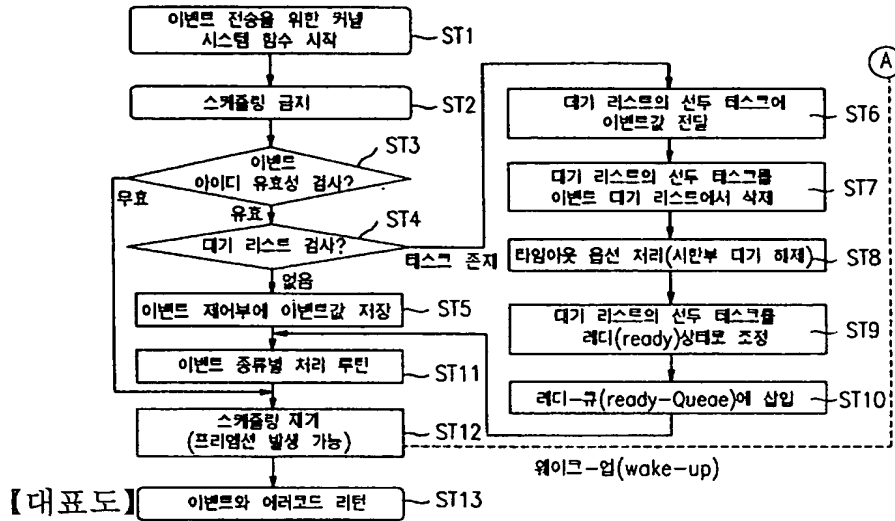
**【첨부서류】**

1. 요약서·명세서(도면)\_1통

**【요약서】****【요약】**

실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법은 실시간 특성이 요구되는 멀티-태스킹 환경에서 높은 우선순위(priority)를 갖는 태스크(task)가 우선적으로 이벤트를 획득하여 실행되도록 하기 위한 것으로서, 우선순위 기반의 프리엠티브 스케줄링(priority-based preemptive scheduling)이 적용되는 상기 멀티-태스킹 환경에서 하나의 이벤트에 대해 그것을 전송받고자 하는 다수개의 태스크들이 각기 이벤트 획득을 위한 커널 시스템 함수(system call)를 호출할 경우 상기 실시간 운영체제 커널이 상기 태스크들을 우선순위(priority) 순으로 이벤트의 대기 리스트(waiting list)에 미리 삽입해 놓는 단계와, 상기 이벤트가 발생할 경우 대기 리스트(waiting)에서 가장 높은 우선순위(priority)를 가진 태스크가 즉시 상기 이벤트를 획득하고 웨이크-업(wake-up)되어 실행되는 단계로 이루어지는데 그 요지가 있다.

**【대표도】**



【색인어】

이벤트전달체계구현

**【명세서】****【발명의 명칭】**

실시간 운영체제 커널(Real-time operating system Kernel)의 이벤트 전달 체계 구현방법{method for implementation of transferring event in real-time operating system kernel}

**【도면의 간단한 설명】**

도 1 은 종래 기술에 따른 운영체제 커널의 일반적인 이벤트 전달체계 구현방법이 적용될 경우 이벤트에 대한 태스크들의 대기 리스트 상태의 한 예를 나타낸 도면

도 2 는 본 발명에 따른 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법이 적용될 경우 이벤트에 대한 태스크들의 대기 리스트 상태의 한 예를 나타낸 도면

도 3a 및 도 3b 는 본 발명에 따른 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법의 일실시예를 구체적인 커널 시스템 함수(kernel system function) 내부 절차로 나타낸 도면

도 4 는 도 2 의 각 태스크의 대기(wait), 웨이크-업(wake-up) 및 실행 상태를 나타낸 도면

도면의 주요부분에 대한 부호의 설명

101 : 제 1 태스크의 이벤트 제어부(Event Control Block)

102 : 제 2 태스크

103 : 제 3 태스크

104 : 제 4 태스크

A : 제 1 태스크 생성 및 실행, 이벤트 제어 생성 시점

B : 제 2 태스크 생성 및 실행 시점

C : 제 3 태스크 생성 및 실행 시점

D : 제 4 태스크 생성 및 실행 시점

S : 제 1 태스크가 이벤트 전송을 위한 커널 시스템 함수를 호출하는 시점들

W : 제 2 내지 제 4 태스크가 이벤트 획득을 위한 커널 시스템 함수를 호출하는 시점들

**【발명의 상세한 설명】**

**【발명의 목적】**

**【발명이 속하는 기술분야 및 그 분야의 종래기술】**

- <15>        본 발명은 실시간 운영체제 커널(Real-time operating system kernel)에 관한 것으로, 특히 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트(event) 전달 체계 구현방법에 관한 것이다.
- <16>        대부분의 운영체제들은 시스템의 효율적인 사용을 위해 다양한 사용자 프로그램(user application program) 태스크(task)를 필요에 따라 동시에 수행하는 멀티-태스킹(multi-tasking) 특성을 지원한다.
- <17>        커널(kernel)은 태스크의 생성과 스케줄링(scheduling), 태스크간 통신(inter-task communication)과 동기화(synchronization), 그리고 기본적인 메모리 관리(memory management), 인터럽트 처리, 디바이스 드라이버 연결체계가 구현된 운영체제의 핵심부로서, 운영체제의 특성과 성능을 좌우한다.
- <18>        실시간 운영체제(Real-time operating system)는 실시간(real-time) 특성이 요구



되는 태스크들의 수행이 필요한 시스템에 적용되는데, 커널은 실시간 태스크(real-time Task)의 일정한 예측시간내(dead-line) 수행을 보장해야 한다.

<19> 이를 위해 실시간 운영체제 커널(Real-time operating system kernel)은 일반적으로 우선순위 기반의(priority based) 프리엠티브 스케줄링(preemptive scheduling) 기법을 쓴다.

<20> 태스크간 통신(inter-task communication)과 동기화는 이러한 스케줄링의 기법과 성능에 있어 중요한 요소가 되는 기능으로, 태스크간의 메시지 전달, 태스크간 상호배제(mutual exclusion)와 동기화(synchronization)를 위한 세마포어(semaphore) 등 이벤트의 전달체제로 구현된다.

<21> 이 이벤트 전달체제는 반응시간(response time)의 최소 및 최적화는 물론이고 우선순위 위주의 프리엠티브 스케줄링을 전제로 하여 설계 및 구현되어야 하는 것이다.

<22> 또한, 실시간 특성을 요하는 시스템이 대부분 내장형 시스템(embedded system)인 점을 감안한다면, 그 시스템에서 구체적으로 요구하는 바에 따라 불필요한 기능과 구조를 배제하고 핵심적인 기능을 보다 간단하면서도 효율적으로 동작하게끔 구현하는 것이 요구된다.

<23> 대기 리스트를 이용하여 이벤트와 그 이벤트를 얻고자 하는 태스크들의 관계를 정의하여 동작시키는 방법이 상기 요구 사항을 만족시키는 해결책이 될 수 있다.

<24> 대기 리스트를 이용하는 종래 기술에 따른 운영체제 커널의 일반적인 이벤트 전달체제 구현방법에 대하여 첨부한 도면을 참조하여 설명하면 다음과 같다.

<25> 도 1 은 종래 기술에 따른 운영체제 커널의 일반적인 이벤트 전달체제 구현방법이 적

용될 경우 이벤트에 대한 태스크들의 대기 리스트 상태의 한 예를 나타낸 도면이다.

<26> 도 1 에 도시된 바와 같이, 이벤트 제어부(1)극 갖고서 이를 통해 이벤트의 발생(전송)을 반복하는 역할의 제 1 태스크와 그 이벤트의 획득을 반복하는 역할의 제 2 내지 제 4 태스크(2~4)의 우선순위(priority)값들은 각각 40, 30, 20, 20(낮은 수치일수록 높은 우선순위)이고, 제 2 내지 제 4 태스크(2~4)는 상기 제 1 태스크의 이벤트 제어부(1)의 이벤트 버퍼(event)에서 값을 얻는다.

<27> 상기 제 1 태스크와 제 1 태스크의 이벤트 제어부(1) 및 제 2 내지 제 4 태스크(2~4)들이 일정 시간 간격을 두고 차례대로 생성되면서 수행된다고 할 때, 제 1 태스크가 생성되어 실행되면서 제 1 태스크는 이벤트 제어부(1)를 생성한다.

<28> 상기 이벤트 제어부는 이벤트의 실체라 볼 수 있는, 커널이 관리하는 자료구조이며, 이벤트 제어부를 생성한다는 것은 태스크 자신이 갖는 이벤트를 만든다는 의미와 같다.

<29> 뒤이어 우선순위가 제 1 태스크보다 높은 제 2 태스크(2)가 생성되면 제 2 태스크(2)가 실행이 되다가 이벤트 획득을 위한 커널 시스템 함수(kernel system function)를 호출하면 아직 제 1 태스크가 이벤트를 하나도 보내지 않은 상태이기 때문에 제 2 태스크(2)는 제 1 태스크가 전송할 이벤트를 기다리기 위해서 대기 상태로 블록(block)되면서 상기 이벤트의 대기 리스트에 연결된다.

<30> 마찬가지로 제 3 및 제 4 태스크(3)(4)도 우선순위가 높지만 이벤트를 기다리기 위한 대기 상태로 블록되어 대기 리스트에 차례대로 연결된다.

<31> 도 1 은 바로 이 순간의 대기 리스트 상태를 나타낸 것이라 할 수 있다.

<32> 이후, 제 1 태스크가 이벤트 전송을 시작하면 제일 먼저 제 2 태스크가 이벤트를 획득

하여 깨어나면서(wake-up) 재실행된다.

- <33> 즉, 피포(FIFO)의 원리에 따라 대기 리스트에 가장 먼저 연결되었던 제 2 태스크(2)에게 먼저 이벤트가 전달되어 제 2 이벤트가 실행될 수 있게 되는 것이다.
- <34> 이는 결국 우선순위(priority)가 상대적으로 더 높은 제 3 또는 제 4 태스크(3)(4)가 우선적으로 실행되지 못하는 결과를 초래한다.
- <35> 즉, 상기의 방법은 일반적인 라운드 로빈(round robin) 방식의 스케줄링에 적용할 수 있지만 실시간 특성이 요구되는 스케줄링 기법을 만족시키지 못할 가능성이 무척 높게 된다.
- <36> 이러한 문제점을 해결하기 위해 대기 리스트의 구조를 이중 연결 리스트(doubly linked list)로 하고 이벤트의 발생 직후 우선순위가 높은 태스크를 대기 리스트에서 찾아내어 우선적으로 수행시키는 방법이 쓰일 수 있다.
- <37> 하지만 이 경우에는 이벤트가 발생(전송)된 이후에 커널 내부에서 대기 리스트를 검색하기 때문에 시간적으로 부담(overhead)이 생기며 이로 인해 반응 시간(response time)이 더디게 되는데, 만약 이벤트에 대기하는 태스크들이 많아진다면 더욱 검색 부담이 커진다.

#### 【발명이 이루고자 하는 기술적 과제】

- <38> 따라서 본 발명은 상기와 같은 문제점을 해결하기 위해 안출한 것으로서, 높은 우선순위를 갖는 태스크가 우선적으로 이벤트를 획득하여 즉시 실행되도록 대기 리스트를 이용하여 간단하면서도 효율적인 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법을 제공하는데 그 목적이 있다.

#### 【발명의 구성 및 작용】

- <39> 상기와 같은 목적을 달성하기 위한 본 발명에 따른 실시간 운영체제 커널(Real-time

operating system kernel)의 이벤트 전달 체계 구현방법의 특징은, 우선순위 기반의 스케줄링(priority-based scheduling)이 적용되는 상기 멀티-태스킹 환경에서 하나의 이벤트에 대해 그것을 전송받고자 하는 다수개의 태스크들이 각기 이벤트 획득을 위한 커널 시스템 함수(kernel system function call)를 호출할 경우 상기 실시간 운영체제 커널이 상기 태스크들을 우선순위(priority) 순으로 이벤트의 대기 리스트(waiting list)에 미리 삽입해 놓는 단계와, 상기 이벤트가 발생할 경우 대기 리스트(waiting list)에서 가장 높은 우선순위(priority)를 가진 태스크가 즉시 상기 이벤트를 획득하고 웨이크-업(wake-up)되어 실행되는 단계를 포함하여 이루어지는데 있다.

<40> 이하, 본 발명에 따른 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법의 바람직한 실시예에 대하여 첨부한 도면을 참조하여 설명하면 다음과 같다.

<41> 도 2 는 본 발명에 따른 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법이 적용될 경우 이벤트에 대한 태스크들의 대기 리스트 상태의 한 예를 나타낸 도면이고, 도 3a 및 도 3b 는 본 발명에 따른 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법의 일실시예를 구체적인 커널 시스템 함수(kernel system function call) 내부 절차로 나타낸 도면이고, 도 4 는 도 2 의 각 태스크의 생성, 대기(wake), 웨이크-업(wake-up) 및 실행 상태를 나타낸 도면이다.

<42> 먼저, 도 2 에 도시된 바와 같이, 제 1 태스크의 우선순위(priority)가 40, 제 2 태스크(102)의 우선순위(priority)가 30, 제 3 및 제 4 태스크(103)(104) 각각의 우선순위가 각각 20이고(수치가 작을수록 높은 우선순위이다), 제 2 내지 제 4 태스크

(102)(104)(103)는 상기 제 1 태스크의 이벤트 제어부(101)를 통해 제 1 태스크가 계속 전송하는 이벤트를 각각 계속 받으려한다고 정의한다.

<43> 도 4 에 도시된 바와 같이, 상기 제 1 태스크가 이벤트 제어부(101)를 생성(A)한 이후, 제 2 내지 제 4 태스크(102)(103)(104)가 일정 시간간격을 두고 차례대로 생성되어 실행될 때(B)(C)(D), 제 2 내지 제 4 태스크(102)(103)(104)는 각각 제 1 태스크의 이벤트 제어부(101)를 통해 이벤트 획득을 위한 커널 시스템 함수(receive function)를 호출하는데, 아직 제 1 태스크가 이벤트를 전송하지 않은 상태이기 때문에 상기 제 2 내지 제 4 태스크(102)(103)(104)는 각각 블록(block)되어 대기 상태가 되면서 상기 제 1 태스크의 이벤트 제어부(101)의 대기 리스트에 연결되게 된다(B')(C')(D').

<44> 상기 제 2 내지 제 4 태스크(102)(103)(104)가 차례로 모두 블록(block)된 시점(도 4 의 E 시점)의 상기 대기 리스트에 연결된 태스크 순은 도 1 에서처럼 커널 시스템 함수(kernel system function call)를 먼저 호출한 순(제 2 내지 제 4 태스크)이 아니라 도 2 의 우선순위 순인 제 3 태스크(103), 제 4 태스크(104), 제 2 태스크(102) 순이 된다.

<45> 즉, 각 태스크마다 이벤트 획득을 위한 커널 시스템 함수(receive function) 호출을 행하여 스스로 대기 상태가 되면서 상기 대기 리스트에 연결될 때, 상기 태스크의 우선순위를 검사하여 대기 리스트가 우선순위 순이 되게끔 상기 태스크를 상기 대기 리스트에 삽입(queueing)함으로써, 대기 리스트를 항상 우선순위가 높은 태스크 순으로 연결된 상태로 관리하는 것이다.

<46> 이후, 도 4 의 F 시점부터 상기 제 1 태스크가 이벤트 전송을 위한 커널 시스템 함수(send function)를 호출하면 상기 대기 리스트의 태스크들에 대한 추가적인 우선순위 검색 없이 바로 상기 대기 리스트의 가장 선두의 태스크를 대기 리스트에서 빼내고(dequeueing)

그 태스크에 이벤트 값을 전달하여 재실행시킨다.

<47> 이는, 대기 리스트는 이미 우선순위 순으로 정렬되어 있는 상태이기 때문에 전송시 대기 리스트의 선두의 태스크를 깨워서(wake-up) 재실행시키기만 하면 결국 우선순위가 가장 높은 태스크가 이벤트를 우선적으로 획득하여 실행되는 결과가 된다는 의미이다.

<48> 결국, 도 4 에 도시된 바와 같이, 제 1 태스크가 전송하는 이벤트는 가장 우선순위가 높은 제 3 태스크(103)와 제 4 태스크(104)가 교대로 획득하여 실행되며, 상대적으로 우선순위가 낮은 제 2 태스크(102)는 이벤트를 획득하지 못한다.

<49> 이와 같이 일실시예에서 사용된 상기 커널 시스템 함수(kernel system function call)들의 내부 절차를 도 3a 및 도 3b 를 참조하여 구체적으로 설명하면 다음과 같다.

<50> 본 발명의 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법은 커널이 제공하는 커널 시스템 함수(kernel system function call)로 구체화되는데, 이는 이벤트 획득을 위한 커널 시스템 함수(wait(또는 pend) function)와 이벤트 전송을 위한 커널 시스템 함수(send(또는 signal) function)의 2가지로 구성된다.

<51> 상기 일실시예에서 설명하였듯이, 각각의 태스크는 모두 자기 이벤트 획득을 위해서 또는 이벤트 전송을 위해서 상기 커널 시스템 함수를 호출하면 상기 시스템 함수의 내부 절차에 따라 상기 일실시예에서 본 바 대로 각각의 태스크가 동작하게 되는 것이다.

<52> 먼저, 태스크가 이벤트를 획득하기 위해 호출하는 커널 시스템 함수(receive function)를 설명하면 다음과 같다.

<53> 현재 수행중인 임의의 한 태스크(현재 태스크)가 이벤트 획득을 위한 커널 시스템 함수(receive function)를 호출했을 때, 먼저 뜻하지 않게 다른 태스크로의 문맥교환(context

switch)이 일어나지 않게 하기 위해 스케줄링을 금지시키고(disable scheduling)(S2) 상기의 현재 태스크가 전달받고자 하는 이벤트의 아이디(event ID)가 유효한지를 검사한다(S3).

<54>        상기 검사 결과 이벤트 아이디가 유효하지 않다면, 존재하지 않는 이벤트에 대해 이벤트 획득을 시도한 오류의 경우이므로, 스케줄링을 재개한 후(enable scheduling)(S12) 오류 코드를 결과로 하여 커널 시스템 함수에서 복귀(return)한다(S13).

<55>        상기 검사 결과 이벤트 아이디가 유효한 경우엔 이벤트 값이 이미 전달되어 있는지를 검사하여(S4), 그 값이 존재하면 이벤트 제어부에서 이벤트 값을 획득하고(S5), 부차적인 이벤트 종류별 처리 루틴을 수행한 후(S11), 스케줄링을 재개한 후(enable scheduling)(S12), 이벤트 값을 갖고 커널 시스템 함수에서 복귀(return)한다(S13).

<56>        즉, 상기 이벤트 값 검사 결과 이벤트 값이 존재한다는 것은 타 태스크가 이벤트를 이미 전송해 놓았다는 사실을 의미하기 때문에, 이벤트 획득을 위한 커널 시스템 함수를 호출한 상기 현재 태스크가 굳이 대기 리스트에 연결되어서 이벤트가 전송되길 기다릴 필요없이 막 바로 상기 이벤트 값을 이벤트 제어부에서 직접 가져가면 된다는 것이다.

<57>        상기 이벤트 값 검사 결과 이벤트 값이 존재하지 않을 경우, 이벤트 획득을 위한 커널 시스템 함수를 호출한 상기의 현재 태스크는 대기 상태(WAIT) 조정이 되고(S6), 이벤트의 대기 리스트(waiting list)에 연결(삽입)이 된다(S7).

<58>        여기서 상기 스텝 7(S7)의 대기 리스트에 커널 시스템 함수를 호출한 상기의 현재 태스크를 삽입하는 과정을 보다 상세하게 기술하면 다음과 같다.

<59>        /\*대기 리스트(waiting list)가 비어있거나(해당 이벤트를 기다리는 태스크가 현재 없

는 경우), 현재 태스크('currentTask')의 우선순위(priority)가 대기 리스트의 후미 태스크의 우선순위보다 낮거나 같으면(대기 리스트에 이미 연결되어 있는 다른 모든 태스크(task)들보다 낮거나 같은 경우) 막바로 대기 리스트의 후미에 연결(삽입)한다.\*/

```
<60>     if waiting list의 선두 task = NULL
<61>         then shortCut = TRUE
<62>         else if currentTask의 priority >= waiting list의 후미 task의 priority
<63>             then shortCut = TRUE
<64>             else shortCut = FALSE
<65>         if shortCut = TRUE
<66>             then currentTask를 waiting list의 후미에 삽입
<67>             return SUCCESS;
<68>     /* 대기 리스트에 1개 이상의 태스크(task)가 이미 있을 경우, 우선순위를 비교하여 대
기 리스트의 태스크들이 우선순위 순이 되도록 현재 태스크를 삽입한다.*/
<69>     prevTask = NULL;
<70>     nextTask = waiting list의 선두 task;
<71>     while(nextTask is not NULL)
<72>     {
<73>         if currentTask의 priority >= nextTask의 priority
<74>             then provTask = nextTask
<75>             nextTask = waiting list에서 nextTask의 다음 task
```



<76>           else currentTask를 prevTask와 nextTask 사이에 삽입

<77>           return SUCCESS;

<78>        }

<79>        return FAIL;

<80>        즉, 상기 스텝 7의 세부 절차는 태스크가 이벤트의 대기 리스트에 연결될 때 상기 대기 리스트의 우선순위 순이 유지되도록 삽입을 하는 과정을 최적화한 것이며, 이로 인해 이후 이벤트 전송을 위한 커널 시스템 함수 호출시 별도의 우선순위 검색없이 가장 높은 우선순위를 갖는 태스크(대기 리스트의 선두 태스크)에게 바로 이벤트를 전달할 수 있게 되는 것이다.

<81>        이어, 상기 현재 태스크가 커널 시스템 함수를 호출할 때 시한부 조건을 주었는지의 여부를 검사하는데, 만일 시한부 대기 시한(timeout 값)이 지정되어 있으면 그 시한이 경과할 때까지 이벤트를 전달받지 못할 경우 이벤트 획득을 위한 대기를 중단하고 깨어나게(wake-up)끔 커널의 타이머 관련 기능을 설정한다(S8).

<82>        이 상태에서 금지시켰던 스케줄링을 재개(enable scheduling)하면(S9), 이는 곧 프리엡션(preemption)을 야기하게 된다.

<83>        현재 수행중이던(커널 시스템 함수를 호출했던) 상기의 현재 태스크는 상기한 바와 같이(S9~S11) 대기 상태로 전환되고 이벤트의 대기 리스트에 연결되어 커널의 스케줄링 대상에서 제외되는 상태이기 때문에 스케줄링이 재개되면 커널은 커널의 레디-큐(Ready-Queue)에 연결되어 있는 준비상태(READY)의 태스크들 중에서 가장 높은 우선순위 등 가장 실행 조건을 잘 만족하는 다른 태스크를 실행시킨다(S10).

<84>        즉, 대기상태(WAIT)에서 수행이 중단되는 상기 현재 태스크와 실행 조건을 만족하는

상기 다른 태스크와의 문맥교환(context switch)이 발생하고, 상기 현재 태스크가 사용하던 자원들을 상기 다른 태스크가 점유하여(preempt) 실행되는 일련의 프리엠션(preemption)이 이루어지는 것이다(S10).

<85>        어느 한 시점에서 어떠한 이벤트의 대기 리스트에 n개의 태스크가 연결되어 있다는 것은 그 n개의 태스크들이 모두 각각 S1~S4 및 S6~S10의 과정을 거쳤다는 의미가 되는데, 상기의 일실시예에서는 제 2 내지 제 4 태스크(102)(103)(104)가 각각 S1~S4 및 S6~S10의 과정을 거치면서 도 2 의 상태가 된 것이다(도 4 의 E 시점).

<86>        스텝 10(S10)에서 대기상태로 머무는 태스크 중 우선순위가 가장 높은 즉, 대기 리스트의 선두 태스크는 이벤트를 보내려는 태스크가 이벤트 전송을 위한 커널 시스템 함수(send function)를 호출하거나 시한부 대기 시한이 경과하면 다시 프리엠션(preemption)이 발생함으로써 비로소 깨워져(wake-up) 재실행되면서 스텝 11 내지 스텝 13(S11~S13)을 거쳐 이벤트 획득을 위한 커널 시스템 함수(receive function)에서 복귀하게 된다.

<87>        한편, 이벤트 전송을 위한 커널 시스템 함수(send function)를 설명하면 다음과 같다.

<88>        현재 수행중인 임의의 한 태스크(현재 태스크)가 이벤트 전송을 위한 커널 시스템 함수(receive function)를 호출했을 때, 우선 이벤트 획득을 위한 커널 시스템 함수(receive function)에서와 마찬가지로 스케줄링을 금지시킨 후(disable scheduling)(ST2) 상기의 현재 태스크가 전달받고자 하는 이벤트의 아이디(event ID)가 유효한지를 검사한다(ST3).

<89>        상기 검사 결과 이벤트 아이디가 유효하지 않다면, 존재하지 않는 이벤트에 대해 이벤트 전송을 시도한 오류의 경우이므로, 스케줄링을 재개시키고(enable scheduling)(ST13)

오류 코드를 결과로 하여 커널 시스템 함수에서 복귀(return)한다(ST14).

<90>       상기 검사 결과 이벤트 아이디가 유효한 경우엔 이벤트의 대기 리스트에 태스크가 존재하는지를 검사하여(ST4), 상기 이벤트에 대해 대기 상태인 태스크가 없다면 이벤트 제어부에 이벤트 값을 저장하고(ST5), 부차적인 이벤트 종류별 처리 루틴을 수행하고(ST11). 스케줄링을 재개한 후(enable scheduling)(St12) 커널 시스템 함수에서 복귀(return)한다(ST13).

<91>       상기 이벤트 저장 과정에 있어서, 이벤트의 종류에 따라 이벤트 제어부(event control block)의 이벤트 버퍼(event buffer)의 상태를 검사하여 이벤트 값의 중복(overwriting) 또는 한계치 초과를 방지해야 하는 경우, 이벤트 버퍼(event buffer)의 넘침(overflow)을 방지해야 하는 경우 등에는 실제 저장과정을 회피하고 오류 코드를 발생시킬 수 있다.

<92>       상기 대기 리스트의 태스크 검사 결과 대기 상태인 태스크가 존재하면, 상기 대기 리스트의 선두 태스크(=대기 리스트에서 우선순위가 가장 높은 태스크)의 이벤트 버퍼(event buffer)에 이벤트 값을 직접 전달(기록)하고(ST6), 상기 선두 태스크를 대기 리스트에서 배내고(dequeueing)(ST7), 시한부 조건이 설정되어 있다면 해제하고(ST8), 상기 선두 태스크를 준비상태(READY)로 조정(ST9)하여 커널의 레디-큐(Ready queue)에 삽입하는(S10) 일련의 과정을 수행하게 된다.

<93>       이때, 상기 이벤트 획득을 위한 커널 시스템 함수(receive function) 수행시 대기 리스트에 태스크들을 우선순위 순으로 연결시켜 놓았기 때문에(S7), 대기 리스트에서 우선순위가 가장 높은 태스크를 검색해야 할 필요가 없다.

<94>       곧이어 스케줄링을 재개하면(ST12) 프리엠션(preemption)의 발생이 가능하게 되는

데 상기 스텝 6 내지 스텝 10(S6~S10)에서 깨어날 준비가 갖춰진 상기 선두 태스크가 재실행될 수 있는 것이다.

#### 【발명의 효과】

- <95>       이상에서 설명한 바와 같이 본 발명에 따른 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법에 있어서는 높은 우선순위(priority)를 갖는 태스크가 우선적으로 이벤트를 획득하여 즉시 실행되도록 함으로써 우선순위 위주 프리엠티브 스케줄링(priority based preemptive scheduling)에 적합한 이벤트 전달 체계를 사용자 프로그램에 제공할 수 있다.
- <96>       또한, 본 발명에 따른 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법은 태스크가 이벤트에 대해 대기하게 될 경우 우선순위(priority)를 검사, 대기 리스트에 삽입하는 선처리 대기 리스트 관리 체계로서, 이벤트가 발생할 경우 반응시간(response time)을 줄이는 등의 성능 개선 효과와 함께 복잡한 요소를 배제한 간결한 구조로 구현되었기 때문에 내장형 시스템(embedded system)용 실시간 운영체제 커널(real-time operating system)에 쉽게 적용할 수 있는 효과가 있다.

**【특허청구범위】****【청구항 1】**

우선순위 기반의 프리엠티브 스케줄링(priority-based preemptive scheduling)이 적용되는 멀티-태스킹 환경에서의 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법에 있어서,

상기 멀티-태스킹 환경에서 하나의 이벤트에 대해 그것을 전송받고자 하는 다수개의 태스크들이 각기 이벤트 획득을 위한 커널 시스템 함수(system call)를 호출할 경우 상기 태스크들을 우선순위(priority) 순으로 이벤트의 대기 리스트(waiting list)에 미리 삽입해 놓는 단계와;

상기 이벤트가 발생할 경우 대기 리스트(waiting list)에서 가장 높은 우선순위(priority)를 가진 태스크가 즉시 상기 이벤트를 획득하고 웨이크-업(wake-up)되어 실행되는 단계로 이루어진 것을 특징으로 하는 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법.

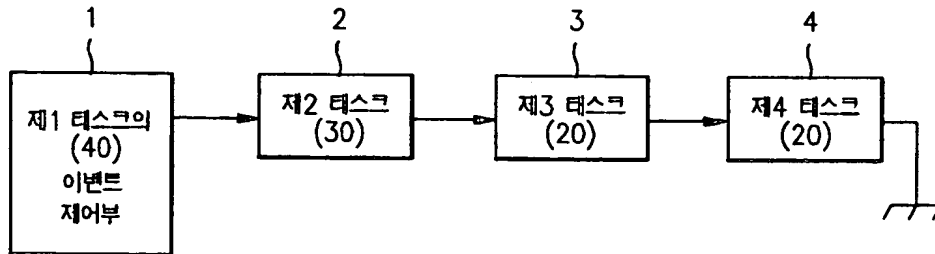
**【청구항 2】**

제 1 항에 있어서,

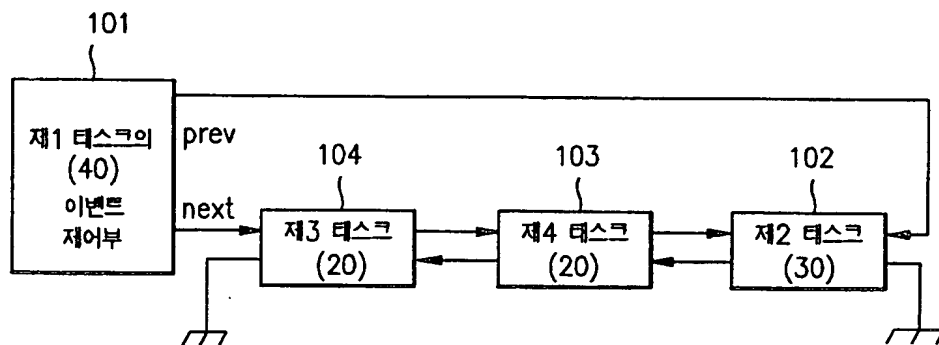
상기 이벤트의 대기 리스트는 대기 리스트의 맨 앞에 가장 높은 우선순위를 가진 태스크가 위치하도록 우선순위 위주로 관리됨을 특징으로 하는 실시간 운영체제 커널(Real-time operating system kernel)의 이벤트 전달 체계 구현방법.

## 【도면】

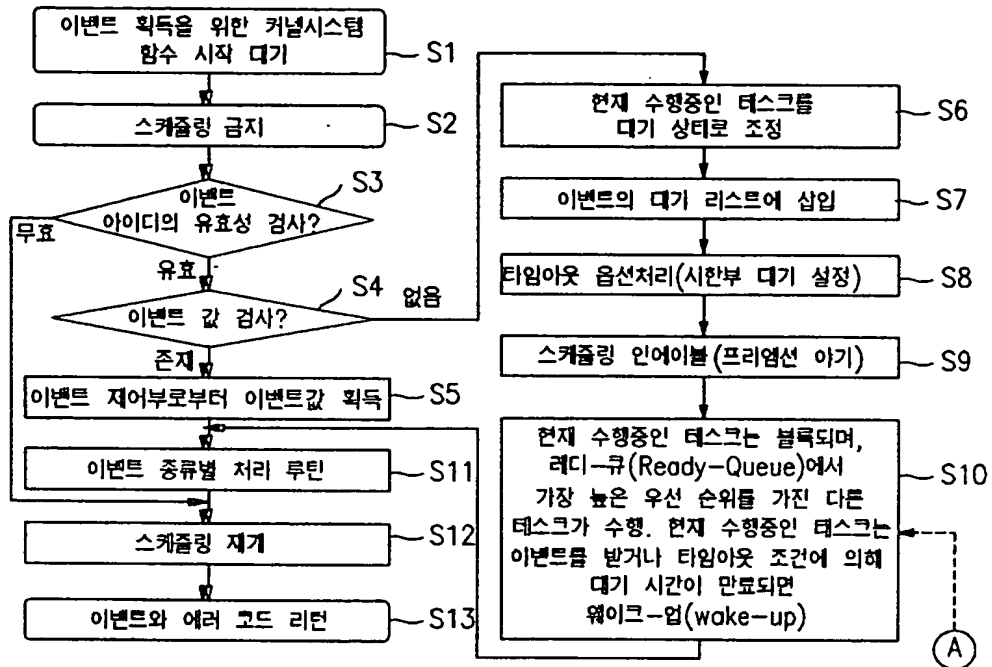
【도 1】



【도 2】



【도 3a】



【도 3b】

